From: "Graunke, Gary" <gary.graunke@intel.com>
To: "'AESFirstRound@nist.gov'" <AESFirstRound@nist.gov>
Subject: performance comments on the 15 AES candidates
Date: Thu, 15 Apr 1999 09:19:36 -0700
X-Mailer: Internet Mail Service (5.5.2448.0)

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
I have attached a preliminary performance study of all 15 AES candidates.
<<Yet Another Performance Analysis of the AES Candidates.doc.asc>>

I have also reproduced the generation of Serpent S-boxes. I believe that all stated criteria were met except the non-linearity for one output of 6 of the 8 S-boxes. This may be an bug on my part, or on the part of the submitter's program, or simply a documentation problem.

```
anf non-linearity for sbox 0:  (0: 3)  (1: 3)  (2: 3)  (3: 2)
anf non-linearity for sbox 1:  (0: 3)  (1: 3)  (2: 2)  (3: 3)
anf non-linearity for sbox 2:  (0: 2)  (1: 3)  (2: 3)  (3: 3)
anf non-linearity for sbox 3:  (0: 3)  (1: 3)  (2: 3)  (3: 3)
anf non-linearity for sbox 4:  (0: 2)  (1: 3)  (2: 3)  (3: 3)
anf non-linearity for sbox 5:  (0: 2)  (1: 3)  (2: 3)  (3: 3)
anf non-linearity for sbox 6:  (0: 3)  (1: 2)  (2: 3)  (3: 3)
anf non-linearity for sbox 7:  (0: 3)  (1: 3)  (2: 3)  (3: 3)
```

Gary Graunke gary.graunke@intel.com
2111 NE 25th AvenueVoice: 503 264 9249
Hillsboro, OR 97124-5961 Fax: 503 264 6225
PGP: 266C B92E 876A 2B1C B30A 225C DCD5 59C8 F2BE DA54

-----BEGIN PGP SIGNATURE-----
Version: PGP 5.5.5

iQA/AwUBNxYQYNzVWcjyvtpUEQJV6wCg0HI3Cji8GodFK4U6EU1lYjS4N+IAn1Va
NadV9ndW+mP7IV051eDEQhq8
=Az8e
-----END PGP SIGNATURE-----

Yet Another Performance Analysis of the AES Candidates

Gary Graunke (gary.graunke@intel.com)

We studied the AES candidate ciphers to get a rough idea of how they might perform on future microprocessors. Since the proposed lifetime of the AES cipher extends well beyond our current design horizon, it was decided to simply arrive at a lower bound for each cipher on an idealized, albeit general purpose computer architecture. This ruled out special purpose instructions that may only be useful to block cipher implementation, but would not be otherwise used in other application areas. In addition, some limitations may be imposed by economics, whether the resource cost becomes exponential as the number of resources is increased.

The target machine operations are micro-operations that have been implemented in the past directly in hardware in one clock cycle, with the exceptions of fetching data from cache/memory, and multiplication. These last two operations were assigned a latency of four clock cycles. We assumed that adequate atomic ALU data widths were supported, up to 64 bits, at the ideal latency. We expect full pipelining of the fetch and multiply operations. Multiple operations can be issued in a vector, VLIW, or data-flow manner. Since the kernels are for the most part branch-free, we did not have to make any assumptions with regard to branch prediction or speculative execution. There are no assumptions of instruction decode problems, instruction encoding, register limitations, or other factors that might make the actual performance exceed the lower bound on many of today's existing architectures.

The implementation strategy and micro-operations were manually selected. The basis of most algorithms owes much to the optimized C language implementations provided by the algorithm submitters, as well as the C language implementations of Brian Gladman et al. We implicitly assumed that only one memory fetch unit would be available, so reduction operations were coded linearly rather than in a reduction-tree form.

We only examined encryption due to resource constraints. We note that CBC decryption can add a significant amount of parallelism to boost performance, whereas CBC encryption is intrinsically serial. We also did not examine key set up times, and assumed that any reasonable amount of work that enhanced encryption at the expense of key setup would be done. A program was written for each cipher to generate the code for the entire encryption of one block, round by round, and compute the metrics.

Finally, the resulting pseudo-code was not tested by simulation to ensure correctness, so there is some chance of error compared with compiled code. However, for some ciphers, some of the scheduled pseudo-code was used to construct actual implementations on existing architectures, and these were tested using the supplied data sets.

We propose two metrics of performance. The first is the critical path of encrypting one block in clocks. The second is the total number of micro-operations that are required, counting vector operations as a single micro-operation. The critical path is a property of the algorithm that cannot be enhanced by any amount of hardware. The operation count provides a better measure of performance for low-end machines. Dividing the operation count by the number of functional units (ignoring asymmetries that might exist) provides a lower bound for mid-range machines.

The results are summarized in table 1 for the 128-bit key and 128-bit block size.

Cipher	Critical Path (in clocks)	Micro-operations
Rijndael	86	616
Crypton	102	736
E2	120	880
Twofish	166	600
RC6	185	296
DFC	232	320
MARS	258	528
Safer+	265	669
Cast256	316	824
HPC	386	260
Loki97	500	1140
Serpent	556	1328
Magenta	745	6892
Deal	1632	3576
Frog	1796	1562

Table 1

Several cipher implementations warrant specific notes.

The Serpent implementation used Brian Gladman’s realization of the 4x4 S-boxes as bit-slice operations. No effort was made to minimize the critical path of the S-boxes. These may be more tuned to reducing operation count. With 18 rounds, the critical path is 311 clocks, and the cipher uses 751 operations. We did not assume a specialized 4X4 S-box instruction, which may be feasible to implement in practice.

The Twofish version did not have pre-compiled keys—fixed code was used.

The DFC implementation assumed a 64X64->128 bit integer multiply. We note that this is generally useful, and also useful for public key algorithms using modular arithmetic. Many existing implementations use only 16 or 32-bit multiplies.

The Safer+ pseudo-code used vector operations, except for the S-box memory fetch operations.

HPC was assumed to have the appropriate 64-bit operations.

Deal was not implemented—we simply multiplied the DES numbers by 6. If the initial and final permutations were removed, DES would get a critical path of 228, and an operation count of 532.

Frog may benefit somewhat as compared with the reported critical path number if hardware were constructed to speculatively execute in spite of potential load/store memory conflicts. Otherwise, the data-dependent, byte-oriented memory accesses are problematic on higher-end machines.

In summary, we found somewhat optimistic lower bounds for each of the candidate ciphers where computer architecture and compilation effects were minimized. Actual implementations that have been reported for current architectures and compilers differ materially for many ciphers. For some, such as DFC, Safer+, and HPC, there is an obvious explanation. For others, the case is less clear. Nevertheless, we believe it is useful to balance highly tuned implementations on existing architectures with the limitations imposed by the cipher itself on long-term performance when evaluating software cipher performance.